Accord

Алгоритм управления распределёнными транзакциями

Konstantin Osipov, kostja@scylladb.com

HighLoad++ 2022

Agenda

1. The evolution to Accord
2. Key ideas
3. Impact

The problem of transaction control

Make concurrent execution look like sequential

- Sequential to a client in isolation: serializability
- To external observer: strict serializability

Typically:

- Operates on top of a **durable** key/value store
- Focuses on A, C and I (atomicity, consistency and isolation) in ACID

Specific example: Alice and Bob

- Alice and Bob work with the same checking account
- Initial balance $100
- Bob withdraws $20
- Alice deposits $10

A schedule of concurrent reads and writes

Alice reads balance, $B_{Alice} = 100$

Bob reads balance, $B_{Bob} = 100$

Alice writes new balance,

$B_{Alice} = 100 + 20 = 120$

Bob writes new balance,

$B_{Bob} = 100 - 10 = 90$!🦆!

The world is more advanced:

1) **The semantics** of the command can be more intricate (think UPSERT, or ReadShared)
2) **The scope** of the command can be big/small (think Cell, Row, Database)

The **schedule**: $T_1R(x),T_2R(x)T_1W(x),T_2W(x)$

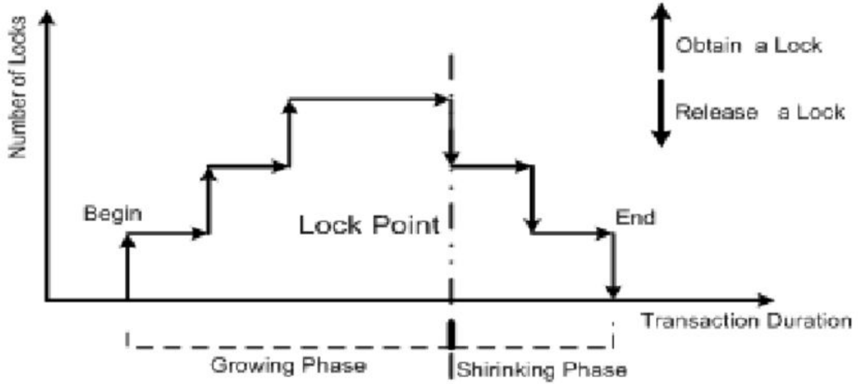
Specific challenges

- **Safety**: Phantoms:
  - **G0, G1a, G1b, G1c, OTV, PMP, P4, G-single, G2-item, G2**
  - AKA dirty writes, dirty reads, observed transaction vanishes, predicate-many-preceders, etc
  - https://github.com/ept/hermitage

- **Liveness:** deadlocks, starvation

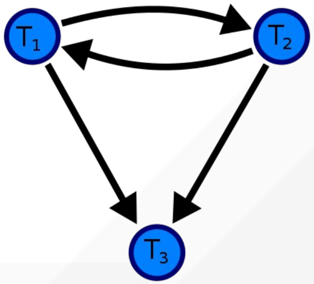
Classical answer: 2PL (don't mix up with 2PC :)

- Acquire locks
- Release when done
- 2PL theorem proves 2PL correctness by transaction precedence



Distributed 2PL issues:

- Cost of coordination
- Distributed deadlock detection



Interactive vs non-interactive transactions

Classical aka **non-deterministic** transactions are **open-ended**:

```java
public void saveStudent(Student student) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        // start a transaction
        transaction = session.beginTransaction();
        // save the student object
        session.save(student);
        // commit transaction
        transaction.commit();
    } catch (Exception e) {
        if (transaction != null) {
            transaction.rollback();
        }
        e.printStackTrace();
    }
}
```

start a transaction

commit a trasaction

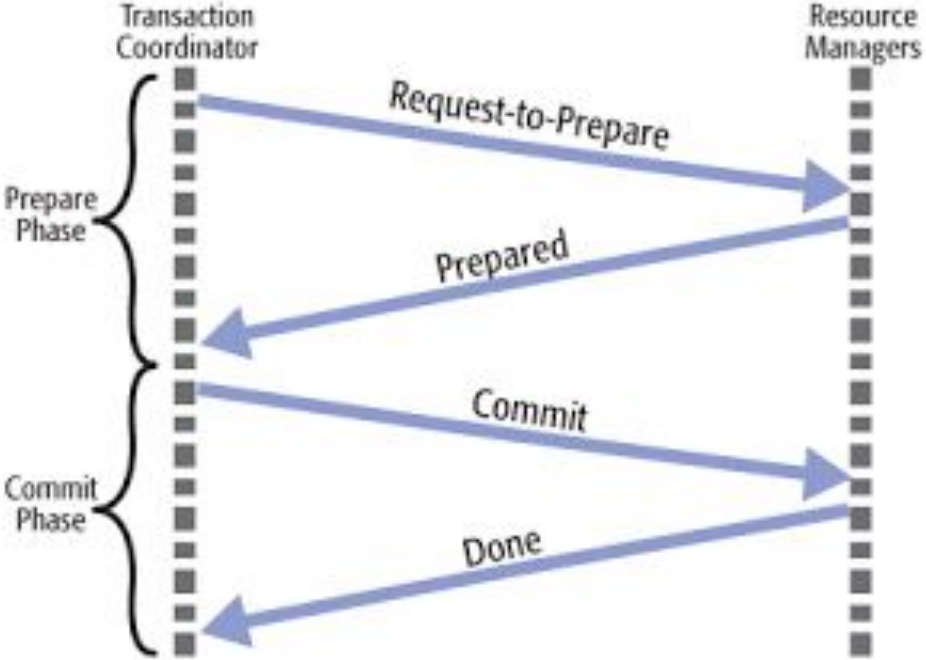
rollback trasaction

Interactive vs non-interactive transactions (2)

Deterministic transactions are fully specified upfront:

```
> BEGIN BATCH
    > UPDATE tasks SET n_abandoned = 0 WHERE project_id  = 1
        > IF n_abandoned > 0
    > DELETE FROM tasks WHERE project_id = 1
        > AND state = 'Abandoned'
> APPLY BATCH;
```

Interactive transactions with multiple participants: 2PC



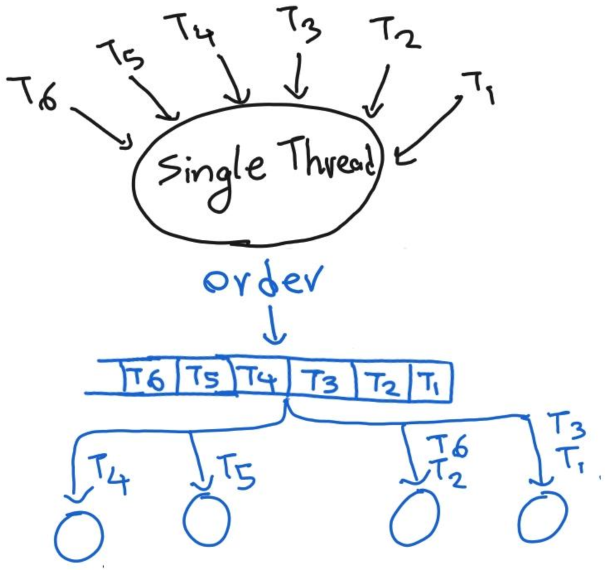
Interactive transactions beget challenges

Why 2PC could roll back?

- Timeouts
- Coordinator failures
- Logical conflicts

Idea: pre-conditions, sequencing

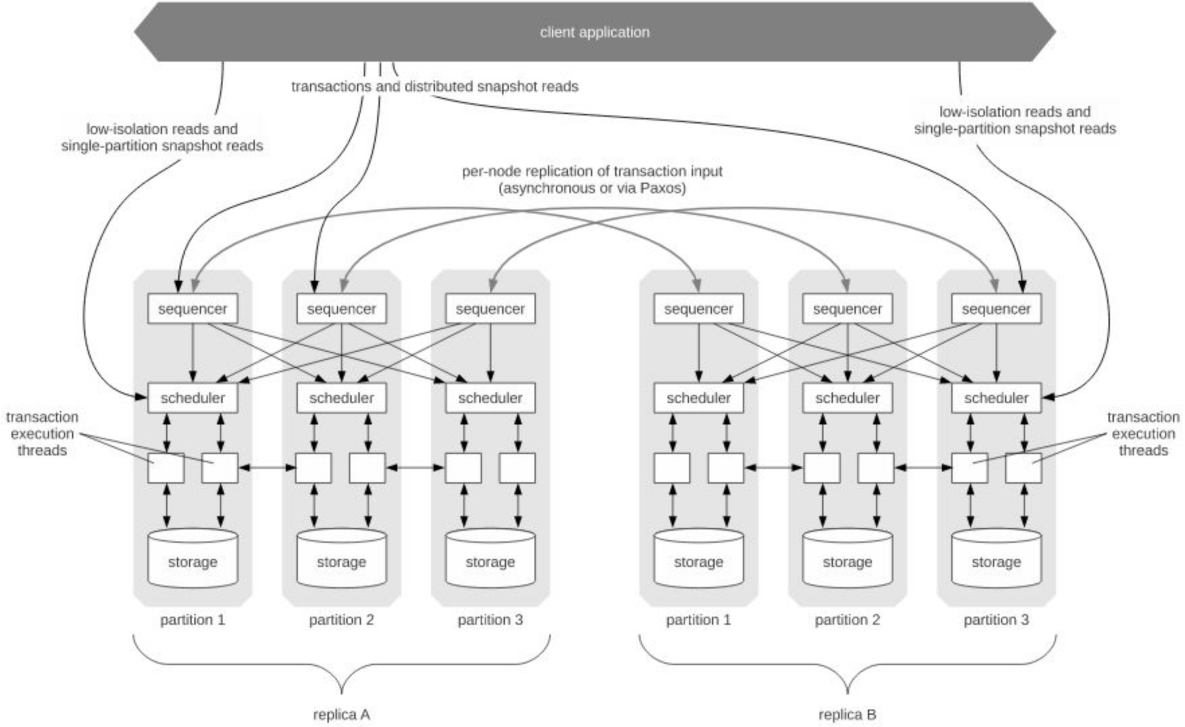
- Record trace of execution (less obvious: reads of secondary index)
- Convert transaction into a batch, trace attached
- If actual trace doesn't match the recorded one, execution is a no-op
- Retry



Sequencing: challenges

- retries affect liveness on hot partitions
- Traces can be long for long transactions
- Double execution can be expensive if reads are costly
- **Sharding the sequencer**

Google Spanner, FaunaDB: Calvin



R2D2 vs C3PO: BigTable vs Spanner

- Google Spanner takes the distributed 2PL approach
- Calvin, AKA BigTable, relies on determinism

… **Accord** is a development on the Calvin route

Deterministic transactions in Cassandra 5.x

```
BEGIN TRANSACTION
   LET a = (SELECT * FROM ….);
   IF a IS NOT NULL THEN
     UPDATE …;
   END IF
   INSERT INTO ...
COMMIT TRANSACTION
```

Deterministic algorithm: deciding on a Sequencer

- Calvin: leader-based, choose once
  - Doesn't scale - needs sharding
    - YDB: sequencer <-> mediator <-> tablet
  - Failure can be expensive
  - Even with sharding, CAS register = entire database
- Tempo: leader-less (aka Paxos)
  - Idea: arrange proposers within a single ballot
  - Attach dependencies to the proposal

Decentralized sequencing

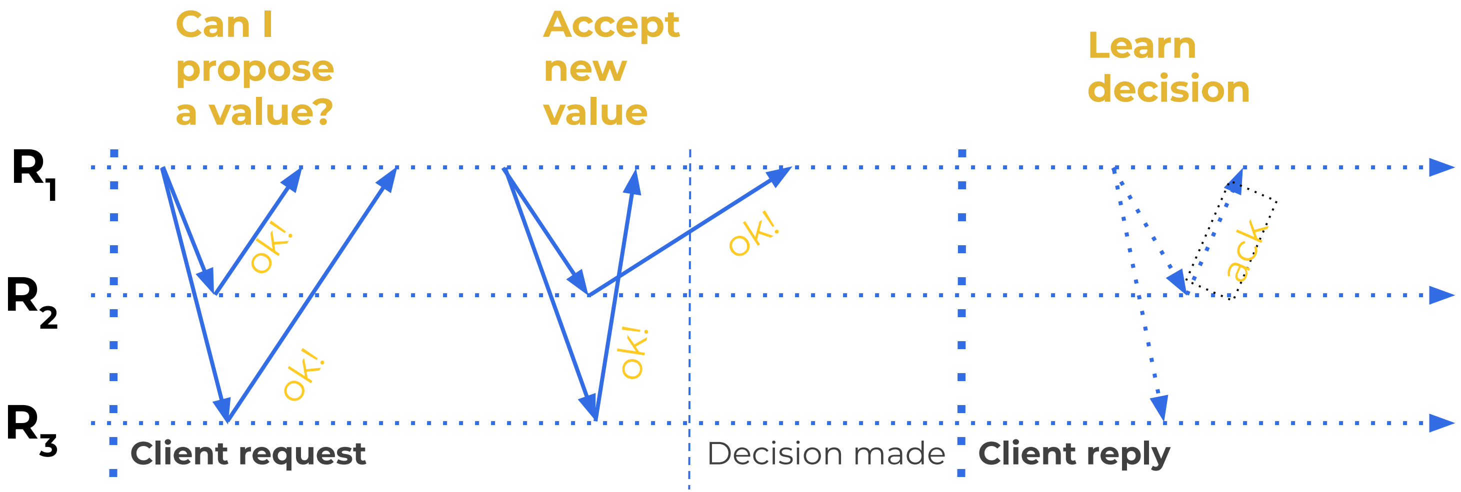
Issues with Paxos:

- too many cross-DC rounds
- liveness under contention

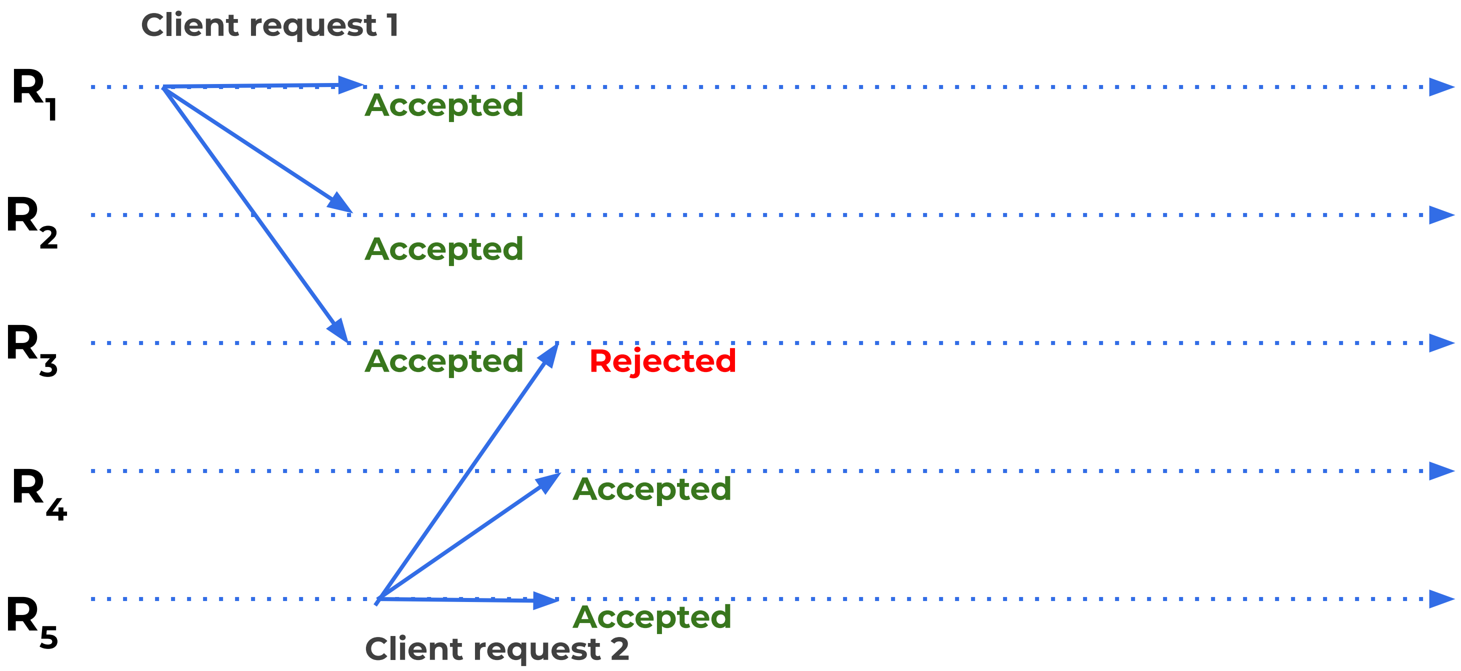
Issues with EPaxos:

- liveness for hot keys

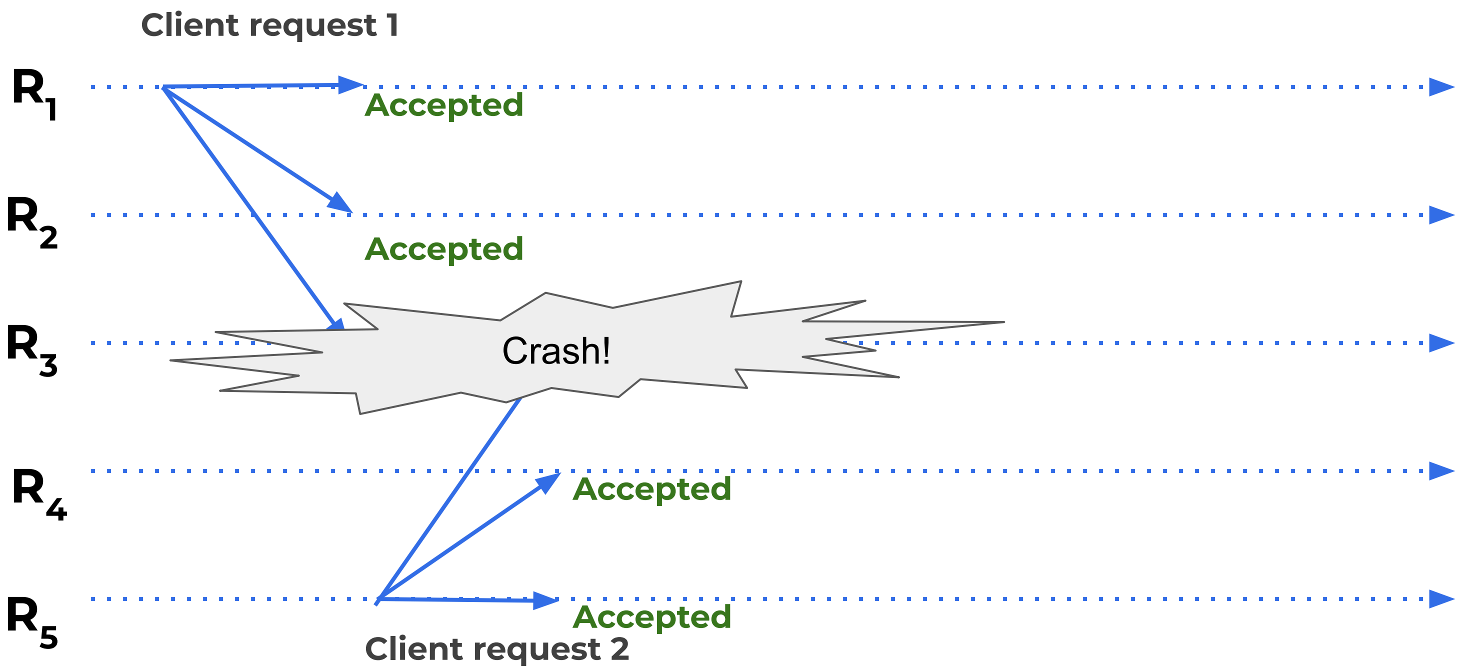
Building up to Accord: basic Paxos



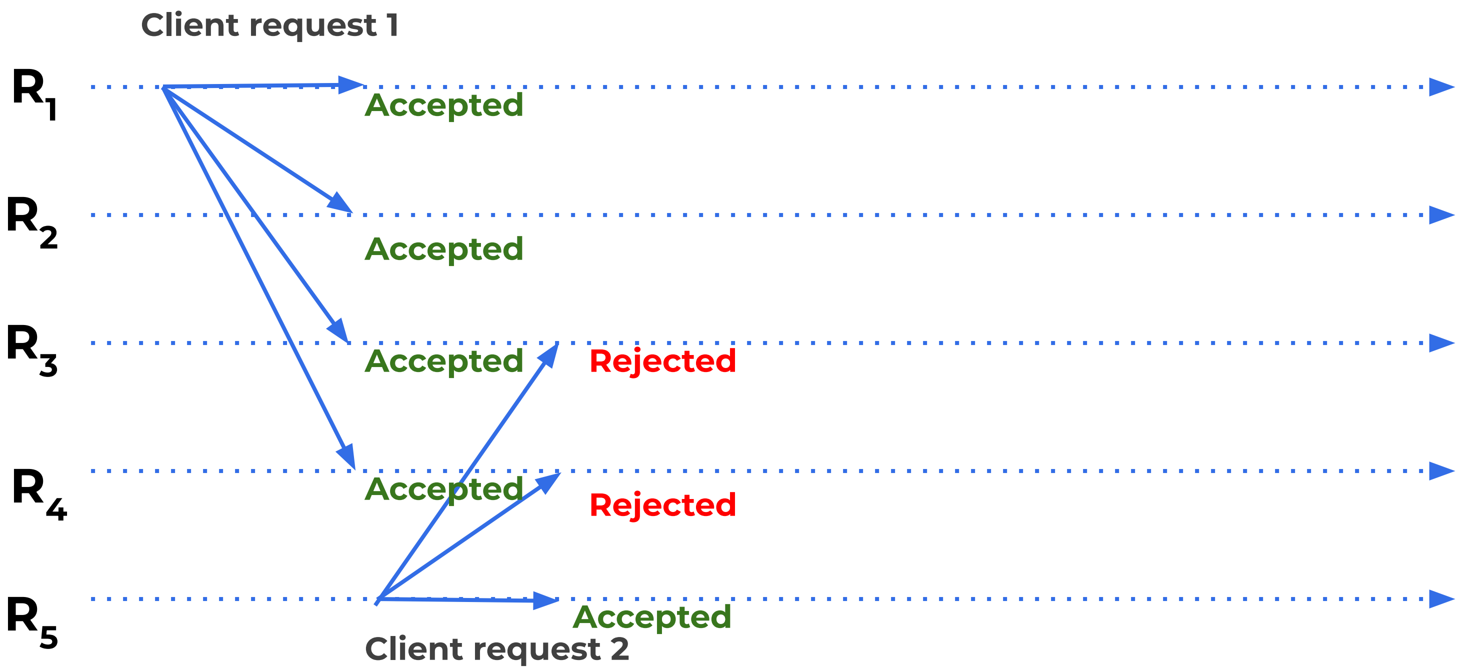
Why does Paxos have two rounds?



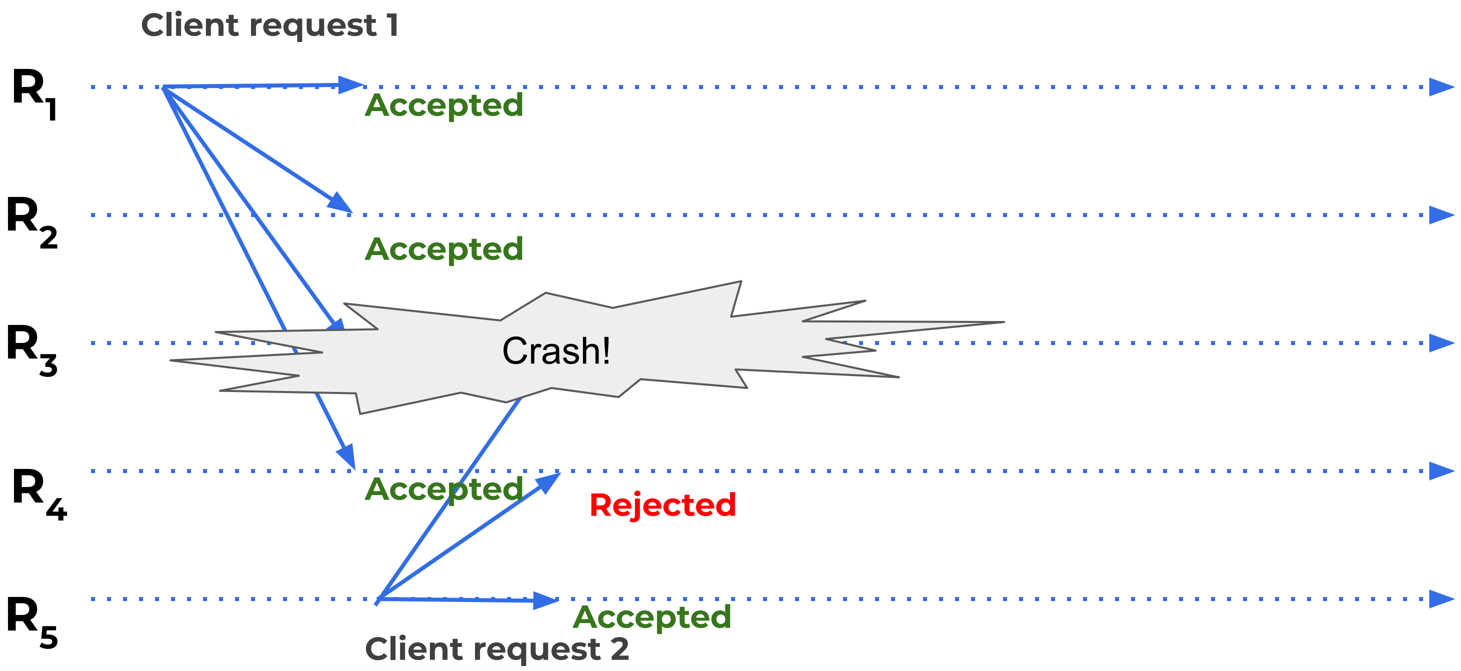
Paxos recovery



Fast paxos: ¾ quorum



Fast Paxos recovery



Fast Paxos quorum intersection Rule

- Speedy completion of a fast quorum (¾ + 1)
- ¾ majority allows to simplify recovery
- We can short-cut the accept round, Propose is enough
- Principle: ¾ + 1 quorum **intersection intersects** with any simple quorum

F_i, F_j – any two fast quorums

Q – recovery quorum

$$F_i \cap F_j \cap Q \neq \varnothing$$

Fast Paxos: recap

- Choose larger quorum (4 out of 5 servers)
- Perform single RTT request & response
    - send transaction to all 5 servers and solicit responses
- Inspect any quorum of responses
    - No collision: quorum containing single accepted value
        - transaction succeeded
    - Collision recovery case I: multiple accepted values w/o majority
        - treat as clean slate
    - Collision recovery case II: multiple accepted values w/ majority
        - run Classic Paxos with the majority value

Avoiding contention: the problem

Alice balance: $100 (= X)

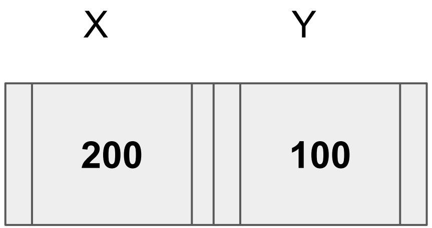
Bob's balance: $200 (= Y)

Alice wants to transfer $10 to Bob

Bob wants to transfer $100 to Alice

$T_1R(x), T_1R(y), T_1W(y), T_1W(x)$

$T_2R(x), T_2R(y), T_2W(x), T_2W(y)$



Valid schedules

$T_1R(x),T_1R(y),T_1W(y),T_1W(x)$ $T_2R(x),T_2R(y),T_2W(x),T_2W(y)$

$\Downarrow$

$T_1R(x),T_1R(y),T_1W(y),T_1W(x),T_2R(x),T_2R(y),T_2W(x),T_2W(y)$

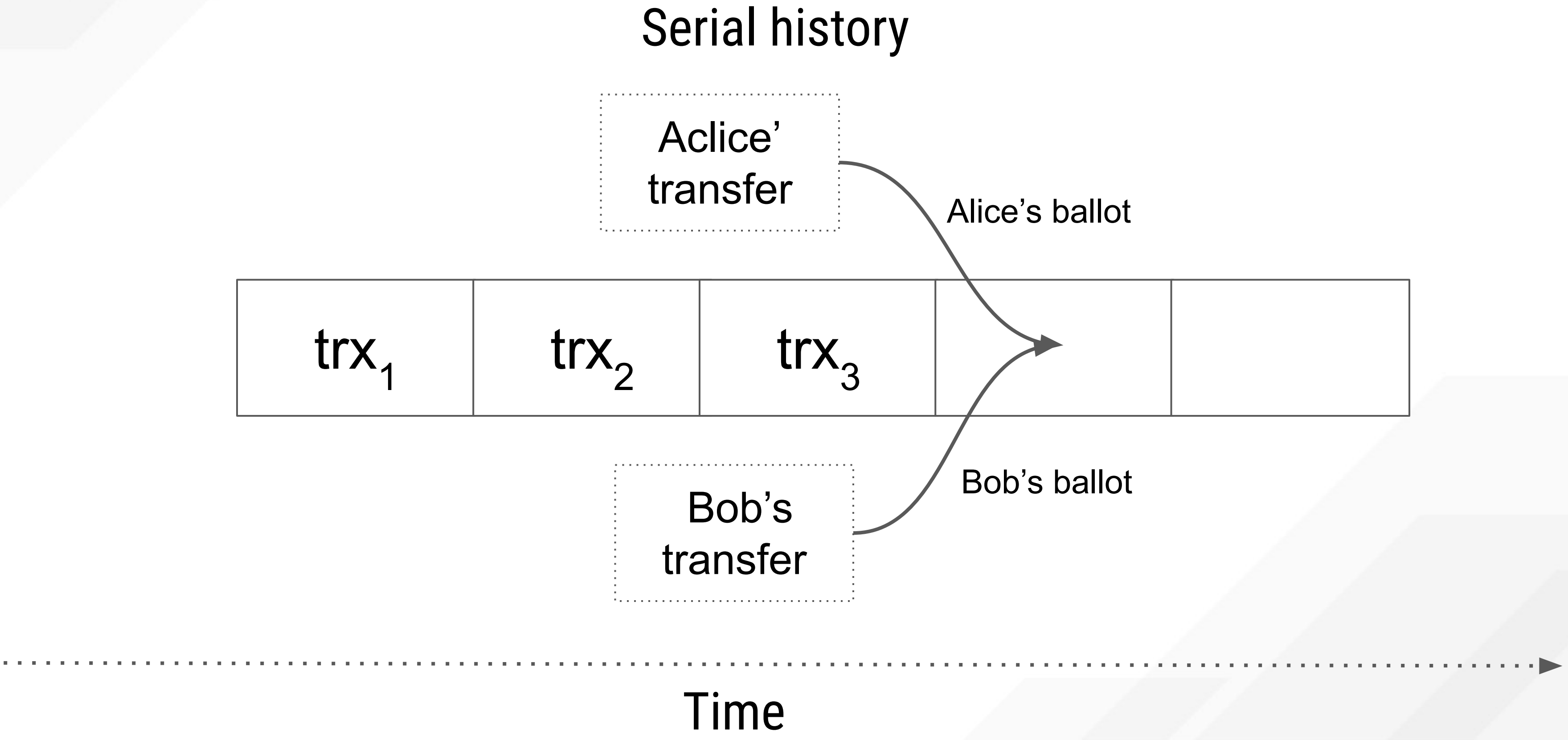
$T_2R(x),T_2R(y),T_2W(x),T_2W(y),T_1R(x),T_1R(y),T_1W(y),T_1W(x)$

~~$T_1R(x),T_1R(y),T_2R(x),T_2R(y),T_1W(y),T_1W(x),T_2W(x),T_2W(y)$~~

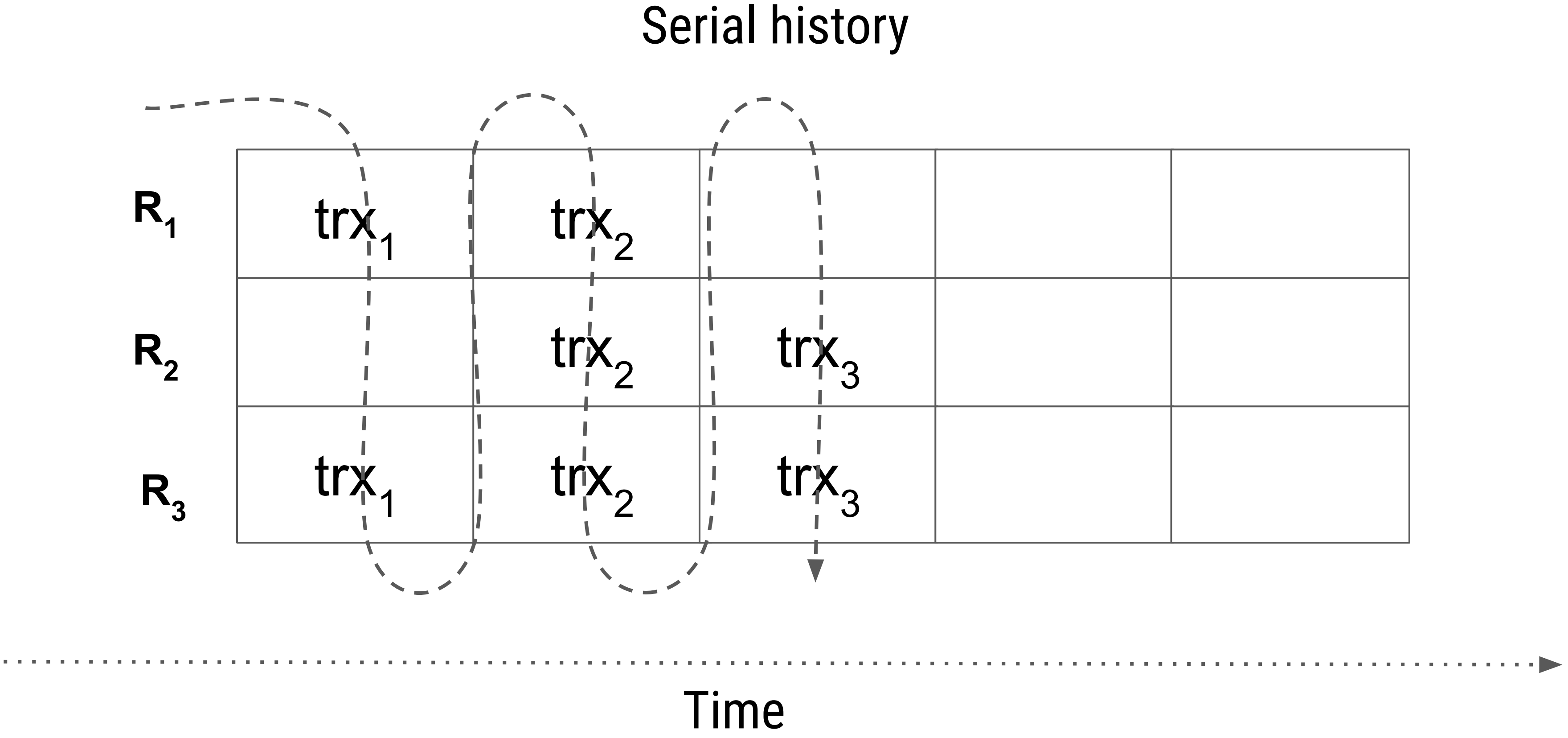
~~$T_2R(x),T_2R(y),T_1R(x),T_1R(y),T_1W(y),T_1W(x),T_2W(x),T_2W(y)$~~

~~$T_2R(x),T_1R(x),T_2R(y),T_1R(y),T_1W(y),T_2W(x),T_1W(x),T_2W(y)$~~

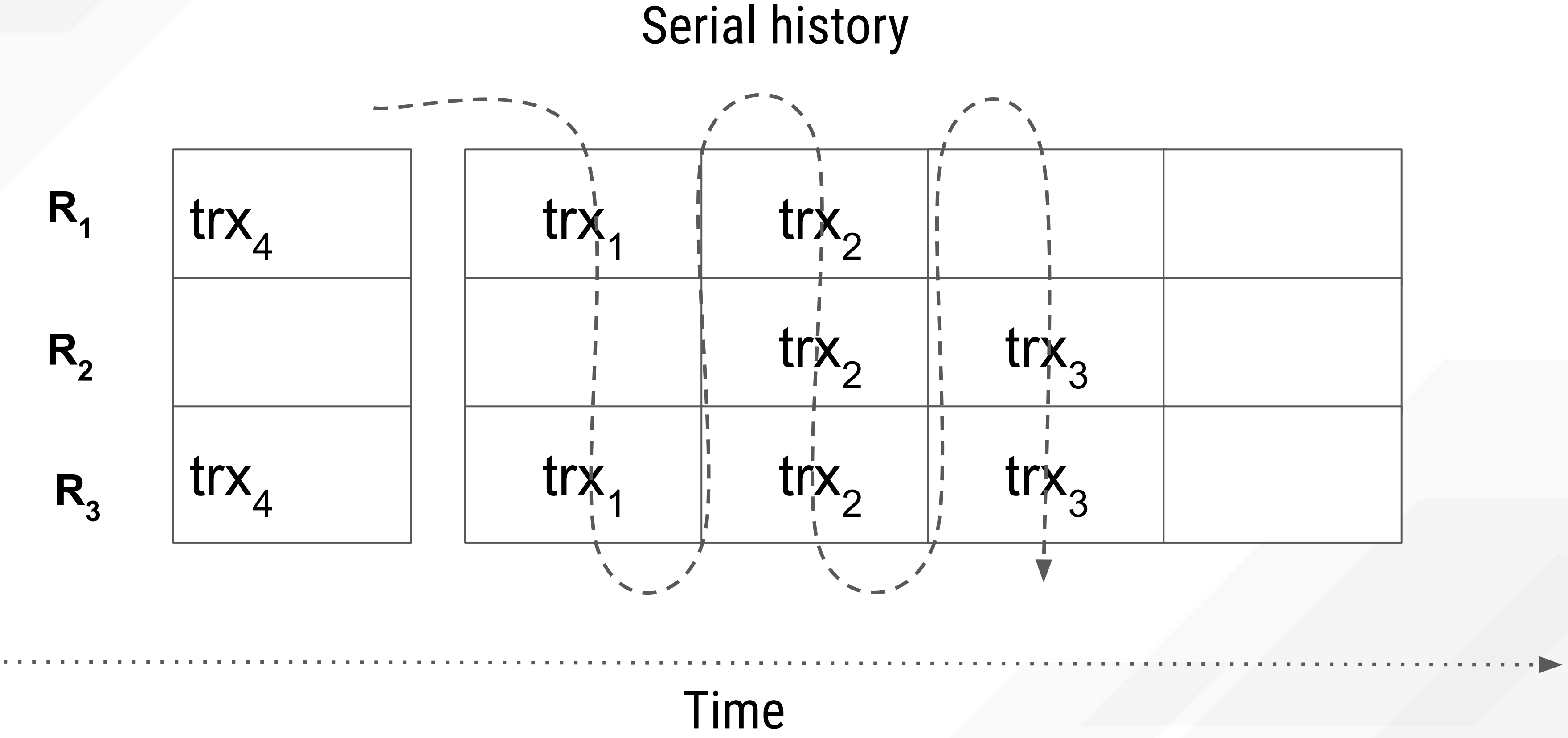
Contention for the next ballot: classic Paxos



Contention for the next ballot: EPaxos



Accord idea: reorder buffer

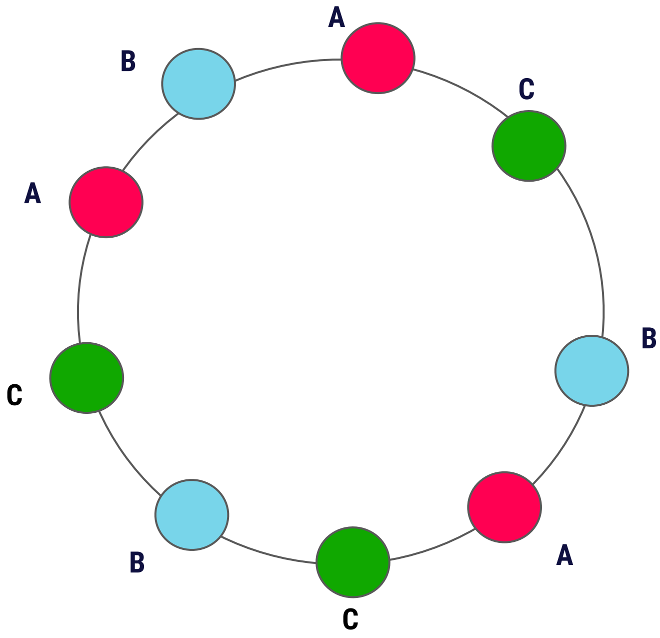


Accord: quorum rules

$T_1 = W(A), W(B)$

$\mathcal{F}_1 = \mathcal{F}_{red} \wedge \mathcal{F}_{blue}$

$\mathcal{Q}_1 = \mathcal{Q}_{red} \wedge \mathcal{Q}_{blue}$



$T_2 = R(B), W(C)$

$\mathcal{F}_2 = \mathcal{F}_{blue} \wedge \mathcal{F}_{green}$

$\mathcal{Q}_2 = \mathcal{Q}_{blue} \wedge \mathcal{Q}_{green}$

Summary: key ideas

- Declare the set of keys upfront (deterministic)
- Reorder buffer to avoid same-key contention
- Track dependencies to avoid ballot contention
- Flexible quorums to reduce Paxos cross-DC cost

Rounds of Accord:

PreAccept -> {Accept} -> Commit -> Read/Execute/Apply

Accord: pros & cons

Pros:

- strict serializable
- works over eventual consistency - same as LWT
- **can be cheaper** than Raft-based transactions in planet-scale set ups
- higher resilience to failure

Cons:

- only deterministic txns - **not solving secondary key/materialized view**
- not implemented yet
- tracking deps can be expensive
- 3 round-trips can be high for single-partition TX

Спасибо!

@kostja_osipov

HighLoad++ 2022

Accord: status

- https://cwiki.apache.org/confluence/display/CASSANDRA/CEP-15
- https://github.com/apache/cassandra-accord
- A library has been in-progress for the last couple of years
- Said to be on track for 5.x
- Supersedes LWT completely: faster and more powerful, including for a single partition